Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/23BE21Perisic

SISTEMI ZA GENERISANJE PROGRAMSKOG KODA

PROGRAM CODE GENERATION SYSTEMS

Ana Perišić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu prikazan je značaj i primena generisanja koda. Opisane su tehnike koje se koriste za generisanje koda. Navedeni su neki od alata za generisanje koda, kao i problemi koji su mogući u slučaju generisanja koda.*

Ključne reči: *generisanje koda, tehnike za generisanje koda, alati za generisanje koda, generatori koda*

Abstract – *This paper describes significance and usage of code generation. Code generation techniques are described as well as code generation tools. Also, some problems, that code generation causes, are listed and described in this paper.*

Keywords: *code generation, code generation techniques, code generation tools, code generators*

1. UVOD

U računarstvu, generisanje koda označava softverske tehnike ili sisteme koji generišu programski kod koji se zatim može koristiti nezavisno od sistema generatora u okruženju koje se izvršava. Osnovna četiri razloga za generisanje koda su: produktivnost, pojednostavljenje, prenosivost i doslednost [1]. Cilj ovog rada je opisati tehnike za generisanje koda, prikazati primenu generatora koda i navesti vrste alata za generisanje koda, kao i neke od problema pri generisanju koda.

U slučaju generisanja koda, generator se piše jednom i može se ponovo koristiti onoliko puta koliko je neophodno. Obezbeđivanje specifičnih ulaza u generator i njegovo pozivanje je znatno brže od ručnog pisanja koda, stoga generisanje koda omogućava uštedu vremena [1].

Sa generisanjem koda se generiše kod iz nekog apstraktnog opisa. Taj opis je obično lakše analizirati i proveriti u poređenju sa celim generisanim kodom [1].

Kada se dobije proces za generisanje koda za određeni jezik ili okvir (engl. *framework*), može se jednostavno promeniti generator i ciljati drugi jezik ili okvir. Takođe moguće je ciljati više platformi odjednom. Na primer, pomoću generatora parsera može se dobiti parser u C#, Javi i C++ [1]. Dakle, isti apstraktni opis se može koristiti za generisanje različitih vrsta artefakata.

Sa generisanjem koda uvek se dobija kod koji je očekivan. Generisani kod je dizajniran prema istim principima. Kod uvek radi onako kako se i očekuje, naravno ako se izuzme pojava grešaka u generatoru. Kvalitet koda je dosledan.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bila dr Dunja Vrbaški, docent.

2. TEHNIKE ZA GENERISANJE KODA

2.1. Šabloni i filtriranje

Ova tehnika opisuje najjednostavniji način za generisanje koda. Sav kod se već nalazi u šablonima, a filtriranje se koristi kako bi se odabrali delovi koda koji će biti uključeni na izlazu iz generatora [2]. Kod koji treba da se generiše nalazi se u šablonima. Promenljive u šablonima mogu biti vezane za vrednosti iz modela.

Generisanje pomoću šablona i filtriranja je prilično jednostavna tehnika, ali glavni nedostatak je kompleksnost stilova. Iz tog razloga, ovaj pristup je potpuno neprikladan za veće sisteme, posebno ako je specifikacija zasnovana na XMI [2].

2.2. Šabloni i metamodel

Da bi se rešio i izbegao problem direktnog generisanja koda iz XML modela, može se implementirati generator na više nivoa koji prvo analizira XML, zatim instancira metamodel, koji je prilagodljiv od strane korisnika, i na kraju ga koristi zajedno sa šablonima za generisanje. Ulaz se koristi za kreiranje metamodela, a potom se dati metamodel koristi zajedno sa šablonima kako bi se na osnovu njega generisao izlaz.

Prednost ovog pristupa je što se s jedne strane dobija veća nezavisnost od konkretne sintakse modela, na primer UML-a i njegovih različitih XMI verzija [2]. S druge strane, može se integrisati funkcionalnija logika za verifikaciju modela, ograničenja, u metamodel [2]. Za razliku od tehnike šablona i filtriranja, ovo se može implementirati u programskom jeziku, kao što je Java.

2.3. Procesiranje frejmova

Frejmovi su, u osnovi, specifikacije koda koje treba generisati. Kao i klase u objektno orijentisanim jezicima, frejmovi se mogu instancirati više puta. Frejm je sličan klasi u objektno orijentisanom programiranju, a instanca objektu date klase.

Tokom instanciranja, promenljive, koje se nazivaju slotovi, su vezane za konkretne vrednosti. Različito ponašanje istog frejma se postiže prosleđivanjem različitih parametara prilikom instanciranja. Svaka instanca može da poseduje sopstvene vrednosti za slotove, baš kao i objekti klasa.

Vrednosti koje su dodeljene slotovima mogu biti različite, od stringova do instanci frejmova. U toku izvršavanja, ovo rezultira strukturom stabla frejmova koja na kraju predstavlja strukturu programa koji treba da se generiše [2].

2.4. Generisanje zasnovano na API specifikaciji

Ubraja se u najpoznatiju tehniku generisanja koda. Kod generisanja zasnovanog na API (engl. *Application Programming Interface*) specifikaciji generisanje se vrši tako što se kreira ciljni model upotrebom namenskog API-ja [4]. Konceptualno se ovo generisanje zasniva na apstraktnoj sintaksi, odnosno metamodelu ciljnog jezika.

Korisniku je dostupan API koji je uprošćena verzija ciljnog jezika i sa njim najčešće deli apstraktnu sintaksu [4]. Klijent koristi API koji je baziran na apstraktnoj sintaksi ciljnog jezika i njime kreira elemente izlaznog programa. Elementi se kreiraju na nivou apstraktne sintakse. Stoga nije moguće kreirati sintaksno neispravan kod. Na primer, može da postoji API koji ima funkciju za kreiranje neke klase u izlaznom programu [4]. Toj kreiranoj klasi je moguće dodati metode, atribute i slično. Na kraju se apstraktno sintaksno stablo izlaznog fajla automatizovano pretvara u tekst, odnosno programski kod [4].

Nedostatak je što se često mora pisati ponavljajući kod koji se u nekim drugim tehnikama nalazi u sastavu šablona [2]. Takođe, mana je složenost i imperativni tok zadavanja izgleda izlaznog programa što može prouzrokovati nečitkost kod složenijih transformacija [4]. Ipak prednost ovog pristupa je što se radi na apstraktnom nivou i nije moguće kreirati sintaksno neispravne izlaze.

Međutim, problem sa ovom vrstom generatora je u tome što postoje potencijalno velike količine konstantnog koda, koda koji ne zavisi od modela, koji mora biti programiran umesto da se jednostavno kopira u šablone [2].

2.5. Generisanje koda in-line

Generisanje koda *in-line* se odnosi na slučaj u kojem izvorni kod sadrži konstrukcije koje generišu više izvornog koda ili mašinskog koda tokom kompajliranja ili neke vrste pretprocesiranja [2]. Predstavlja kombinaciju izvornog i generisanog koda. U izvornom kodu se nalaze sekcije koje kompajler ili interpreter koriste za generisanje dodatnog izvršnog koda. Primer bi bili šabloni u C++ programskom jeziku [2].

Ciljni jezik poseduje iskaze koji se u vreme kompajliranja zamenjuju drugim konstrukcijama istog jezika. Ova zamena se obavlja najčešće posebnim alatom koji se naziva pretprocesor. Pretprocesiranje može da se obavlja u više koraka, odnosno moguće je imati više pretprocesora koji će vršiti ekspanziju koda pre faze kompajliranja.

2.6. Generisanje koda atributima

Tehnika koja se često primenjuje u programskom jeziku Java. Postala je popularna zahvaljujući *JavaDoc* u Java programskom jeziku, gde su se specifični komentari koristili da bi omogućili automatsko generisanje HTML dokumentacije [2]. S obzirom na proširivost *JavaDoc* arhitekture, moguće je uključivanje prilagođenih oznaka (engl. *Custom tags*), kao i generatora koda.

Najpoznatiji primer predstavlja *XDoclet*, poznatiji kao XDOC [2]. *XDoclet* omogućava generisanje EJB interfejsa (engl. EJB *Remote/Local Interfaces)* i deskriptora (engl. *Deployment descriptors)*. Programer ručno piše klasu implementacije i dodaje neophodne *XDoclet* komentare u klasu, koje zatim čita *XDoclet* generator koda. Štaviše, generator ima pristup sintaksnom stablu izvornog koda, kome se dodaju komentari. Na ovaj način generator može da izvede informacije iz komentara kao i iz samog koda [2].

2.7. Generisanje zasnovano na aspektnom pristupu

Spada u tehniku koja opisuje spajanje odvojenih, ali sintaksno potpunih i nezavisnih delova koda. Stoga je neophodno definisati način na koji će se različiti delovi koda spojiti, pa se uvodi pojam tačke spajanja (engl. *join points, hooks*). *AspectJ* [ASPJ] je dobro poznati primer ove vrste generatora: regularni OO programski kod i kod aspekta su isprepleteni, bilo na nivou izvornog koda ili na nivou bajt koda [2]. Aspekti opisuju sveobuhvatne probleme, odnosno, funkcionalnost koja se ne može adekvatno opisati i lokalizovati korišćenjem dostupnih konstrukcija objektno orijentisanog programiranja [2].

3. PRIMENA GENERATORA KODA

U zavisnosti od konteksta, generatori koda su našli primenu u različitim oblastima, s obzirom da predstavljaju značajnu komponentu razvojnog procesa. Primer primene generatora koda je njegova upotreba u modernim razvojnim okruženjima, gde omogućavaju kreiranje kostura klase za implementaciju interfejsa, pritisnuvši svega jedno dugme [1]. Na taj način je moguća ušteda vremena onome ko razvija softver.

Postoji više različitih načina kako dizajnirati pajplajn generisanja koda (engl. *pipeline*) [1]. Prvobitno je neophodno definisati dva elementa, kao što su ulazi i izlazi. Ulazi predstavljaju mesto odakle dolaze informacije koje će se koristiti prilikom generisanja koda, dok izlazi predstavljaju način kako će se dobiti izgenerisan kod. Takođe, moguće je primeniti niz transformacionih koraka između ulaza i izlaza, što dovodi do pojednostavljenja izlaznog sloja i nezavisnosti ulaza od izlaza. Međutim, primena ovih transformacija predstavlja opcioni izbor.

U nastavku će biti opisani mogući ulazi [1]. Neki od njih su:

- *DSL jezici (engl. Domain Specific Language)* - moguće je koristiti alate kao što je ANTLR kako bi se opisala gramatika jezika, iz čega je posle moguće generisati parser.
- *Kod u različitim formatima* - šeme baza podataka iz kojih se generiše DAO, obrazac objekta pristupa podacima koji odvaja klijentski interfejs od mehanizma pristupa podacima.
- *Obrnuto inženjerstvo (engl. Reverse engineering)* - informacije se mogu dobiti obradom složenih artefakata koda.
- *Pomoćnik (engl.Wizard)* - oni dozvoljavaju zahtevanje informacije od korisnika.
- Izvori podataka kao što su: DB, CSV datoteka ili tabela.

Mogući izlazi su :

- *Obrađivači šablona* (engl. *Template engines*) - većina veb programera poznaje mehanizme šablona koji se koriste za *popunjavanje* HTML korisničkog interfejsa podacima.
- API-ji za pravljenje koda: na primer, Java parser se može koristiti za programsko kreiranje Java datoteka.

U nastavku će biti proučeni neki od pajplajnova, kao što su:

- *Parser generisanje* - odnosi se na generisanje parsera na osnovu zadate formalne gramatike. Ulaz može biti DSL, a izlaz se generiše upotrebom šablonskih mehanizama.
- *Model vođen dizajnom* (engl. *Model driven design*) - dodaci za razvojna okruženja (engl. *Plugins for IDEs*) ili sama razvojna okruženja, koja omogućavaju da se opiše model aplikacije. Često se koristi grafički interfejs i potom se iz *tog* modela generiše ili cela aplikacija ili kosturi klasa.
- *Metaprogramski jezici* - uključuju jezike koji omogućavaju skoro potpunu manipulaciju kodom programa, izvorni kod je samo *još* jedna struktura podataka kojom se može manipulisati.
- *Kod koji se odnosi na bazu podataka* - slično kao i kod modela vođenog dizajnom i šablonima. Obično programer definiše šemu baze podataka iz koje se mogu generisati čitave CRUD aplikacije ili samo kod za rukovanje bazom podataka. Postoje i alati koji obavljaju obrnuti proces: iz postojeće baze podataka kreiraju *šemu* baze podataka ili kod za rukovanje.
- *Ad-hok aplikacije* - ova kategorija uključuje sve: od alata dizajniranih za rukovanje jednom stvari do sistema koji se koriste u poslovnom okruženju, koji mogu generisati čitave aplikacije iz formalnog prilagođenog opisa. Ove aplikacije su uglavnom deo specifičnog toka posla. Na primer, korisnik koristi grafički interfejs da opiše aplikaciju i jedan ad-hok sistem generiše šemu baze podataka za podršku ovoj aplikaciji, drugi generiše CRUD interfejs i slično.
- *IDE generisani kod* - mnogi jezici zahtevaju mnogo šablonskog koda za pisanje i IDE obično može da generiše neke od njih: klase sa opisom za metode koje treba implementirati, *hashCode* i *toString* metode, *get* i *set* metode za sva postojeća svojstva i slično.

Generatori se takođe mogu koristiti za proizvodnju prototipova umesto produkcionog koda (engl. *production code*) [3]. Izrada prototipa se obično koristi za dobijanje ranih povratnih informacija, a generatori mogu proizvesti kod za potpuno drugačiju ciljnu platformu i na drugom programskom jeziku od finalnog koda. Ovde generatori ne moraju nužno da optimizuju kod, već su tu da bi omogućili funkcionalnost, upotrebljivost, izgled ili druge karakteristike relevantne za izradu prototipa [3]. Jezik modelovanja, međutim, može biti isti za razvoj i prototipa i proizvodnog koda. Model se takođe može koristiti za simulaciju. Generatori koda onda obezbeđuju potreban izlaz u simulatoru koda [3].

Generatori se mogu koristiti i za izradu dokumentacije [3]. Takođe je vredno napomenuti da se generisana dokumentacija ne odnosi samo na implementaciju već pokriva i rešenje opisano u domenu. Na kraju krajeva, u DSM (engl. *Domain Specific Model*) metodologiji softverskog inženjerstva, modeli uglavnom specificiraju domen problema, a ne rešenje.

U DSM metodologiji, generatori se takođe koriste za proveru konzistentnosti i kompletnosti dizajna. Ovo je neophodno jer obično nema smisla, ili čak nije moguće, staviti sva pravila u metamodel i proveriti ih tokom svake radnje modelovanja. Ovo je posebno tačno prilikom provere delimičnih modela, kada postoji više modela ili kada se integrišu modeli napravljeni od strane različitih programera [3].

Generatori za analizu modela se takođe mogu koristiti za vođenje rada na modelovanju i informisanju o potrebnim akcijama. Tipičan takav primer je da se pogleda da li je model ili su modeli nedovršeni i da se prijave moguće radnje koje su neophodne da bi model bio potpun. Takva provera modela se može pokrenuti slično kao kod generatora: kada je neophodno ili nakon sprovođenja određenih radnji modelovanja.

4. ALATI ZA GENERISANJE KODA

4.1. Primeri alata za generisanje koda

Obrađivači šablona (engl. *Template engine*) predstavljaju grupu alata koja se najčešće koristi. *Template engine* predstavlja mini kompajler koji je u mogućnosti da prevede jednostavne šablone za odgovarajući programski jezik [1]. Postoji fajl za šablon koji sadrži posebne oznake koje se mogu tumačiti upravo pomoću šablona.

Najjednostavnije što može da uradi jeste da zameni ovu specijalnu notaciju odgovarajućim podacima koje dobije tokom izvršavanja. Većina ovih alata omogućava upotrebu naredbi za kontrolu toka, kao što su: *for* petlje, *if-else* iskazi što korisniku dozvoljava opis jednostavnijih struktura.

Neki od predstavnika obrađivača šablona su:

- U Javi: FreeMarker, Velocity, JSP, StringTemplate, i slično.
- U Pajtonu: Jinja, Django template system, Cheetah, Mako, Genshi, i slično.

Parser generatori predstavljaju grupu alata za automatsko i brzo kreiranje parsera za jezik. Na osnovu formalne gramatike koju dobiju na ulazu generišu programski kod parsera koji će prepoznavati rečenice na datom jeziku i zatim pretvarati ulazne stringove u stabla parsiranja. Sem toga, mogu generisati i skenere, odnosno leksere. Često implementiraju mehanizam za obilazak stabla parsiranja i njegovu transformaciju. Primeri poznatijih parser generatora su: ANTLR, JavaCC, Bison i slično.

4.2. Problemi pri korišćenju alata za generisanje koda

Kada se koristi alat za generisanje koda tada kod postaje zavisan od njega. Zaključuje se da je neophodno održavati alate za generisanje koda, tako da je jedan od problema generisanog koda upravo održavanje generatora koda [1]. Neophodno je konstantno ažurirati postojeći kod alata za generisanje koda.

Još jedan od problema generisanog koda jeste njegova kompleksnost. Automatski generisani kod ima tendenciju da bude složeniji od ručno pisanog koda. Generisani kod je takođe sigurno manje optimizovan od onog koji je moguće ručno napisati. Ponekad je razlika mala i nije značajna, ali ako su aplikaciji značajne performanse, generisanje koda možda neće biti optimalno.

Postoje različita mišljenja o tome koje vrste koda se mogu generisati i koji je nivo kvaliteta tog koda [3]. Na primer, automatizacija za proizvodnju statičkih deklarativnih definicija u odnosu na uobičajeni dizajn, kao što su interfejsi ili šeme baza podataka, postoji već godinama, tako da je na raspolaganju više gotovih generatora [3]. Međutim, situacija se razlikuje kada je u pitanju generisanje ponašanja funkcionalnog koda. Često se zahtevaju opsežni i detaljni

UML modeli za specifikaciju funkcionalne strane, ali još uvek nisu adekvatni u detaljima za generisanje koda.

Mnogi programeri su imali loše iskustvo sa dostupnim generatorima (engl. *third-party generators*) jer je proizvođač generatora popravio način proizvodnje koda [3]. Uprkos postojanju više načina za pisanje koda za određeno ponašanje, proizvođač je izabrao samo jedno od njih. Taj izabrani način nije uvek zadovoljavajući i optimalan za neke postojeće specifične zahteve, uzimajući u obzir generisani ciljni jezik, korišćeni model, memoriju i slično. Generatori koda ne znaju dovoljno o specifičnim zahtevima organizacija da bi generisali idealan kod, tako da je isto tako to jedan od razloga zbog čega se smatra nezadovoljavajućim i nedovoljno korektnim. Pošto izmena generisanog koda obično nije optimalna opcija, organizacije odbacuju generisani kod. Vrednost generisanog koda je tada ograničena na izradu prototipa i fazu prikupljanja zahteva [3].

5. ZAKLJUČAK

U radu je prikazana primena i značaj generatora koda, kao i tehnike i alati koji se koriste za generisanje koda. Zaključuje se da generisanje koda obezbeđuje: produktivnost, pojednostavljenje, prenosivost i doslednost, što omogućuje primenu generatora koda u različitim oblastima.

Na primer, generatori koda zauzimaju značajno mesto u izradi dokumentacije, proizvodnji prototipova, u DSM metodologiji, gde proveravaju konzistentnost i kompletnost dizajna.

Takođe, postoje alati za generisanje koda, kao što su parser generatori koji služe da automatski i brzo kreiraju parser za određeni programski jezik.

Postoji još jedna grupa alata, kao što su obrađivači šablona koji omogućavaju prevođenje napisanih šablona od strane programera na željeni programski jezik.

Međutim, ukazano je i na potencijalne probleme sa kojim se moguće suočiti tokom korišćenja alata za generisanje koda. Zaključuje se da su to: kontinuirano održavanje i ažuriranje programskog koda alata za generisanje koda, kompleksnost generisanog koda, zavisnost koda i alata, kao i kvalitet generisanog koda, neki od problema koje generisanje koda može da prouzrokuje.

6. LITERATURA

[1] A Guide to Code Generation, https://tomassetti.me/code-generation/ (pristupljeno u oktobru 2022.)

[2] T. Stahl, M. Völter, J. Bettin, A. Haase, S. Helsen, *Model-Driven Software Development, Technology, Engineering, Management*, Chichester: John Wiley & Sons Ltd, 2006, pp 186-198.

[3] S. Kelly, J. P. Tolvanen, *Domain-Specific Modeling: Enabling Full Code Generation*, New Jersey: John Wiley & Sons Ltd , 2008, pp 79-86.

[4] I. Dejanović, *Jezici specifični za domen*, Novi Sad: Fakultet tehničkih nauka, 2021.

Kratka biografija:

Ana Perišić rođena je u Trebinju 1998. god. Fakultet tehničkih nauka u Novom Sadu, studijski program Računarstvo i automatika, upisala je 2017.god. Diplomirala je 2021.god., a potom upisala master akademske studije iz iste oblasti.

kontakt: anaperisic129@gmail.com